PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : G06F 9/45

A1

(11) International Publication Number: WO 92/03782

(43) International Publication Date: 5 March 1992 (05.03.92)

(21) International Application Number: PCT/US91/04071

(22) International Filing Date: 10 June 1991 (10.06.91)

(30) Priority data:
571,954 23 August 1990 (23.08.90) US

(71) Applicant: SUPER-COMPUTER SYSTEMS LIMITED PARTNERSHIP [US/US]; 1414 W. Hamilton Avenue, Eau Claire, WI 64701 (US).

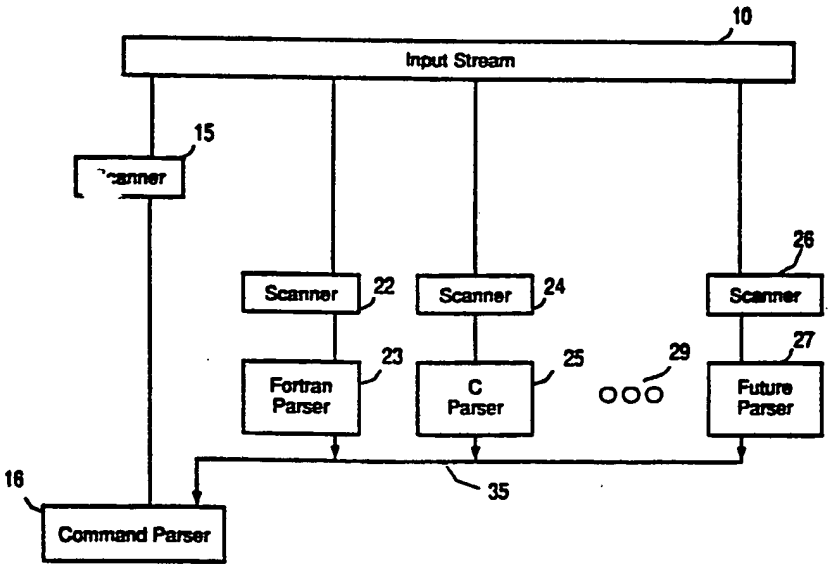
(72) Inventors: CHANDRAMOULI, Ashok ; 40073 Fremont Boulevard, Apartment 611, Fremont, CA 94538 (US). STROUT, Robert, E., II ; 948 Kern Court, Livermore, CA 94550 (US). MASAMITSU, Jon, A. ; 1873 Creek Road, Livermore, CA 94550 (US).

(74) Agents: SIRR, Francis, A. et al.; 3445 Penrose Place, Suite #210, Boulder, CO 80301 (US).

(81) Designated States: AT (European patent), BE (European patent), CH (European patent), DE (European patent), DK (European patent), ES (European patent), FR (European patent), GB (European patent), GR (European patent), IT (European patent), JP, KR, LU (European patent), NL (European patent), SE (European patent).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: PARSING PROGRAM DATA STREAMS



(57) Abstract

An extensible dual-level parsing method consists of multiple parsers including a command language parser (16) and a plurality of programming language parsers (23, 25, 27, 29), such as Fortran (23) and C (25). The command parser (16) parses until it recognizes a language specific item whereupon it calls the appropriate progamming language parser. Each included parser has its own separate parse table and its own lexical scanner (15, 22, 24, 26). All lexical scanners share a common input stream (10). The grammar for each of the parsers is simpler than an all-inclusive grammar covering a command language and various programming language expressions. This simplicity improves p:ing speed and efficiency. The method employs an extensible method of incorporating programming language-specific syntax where necessary within the debugger syntax.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	ES	Spain	MG	Madagascar
AU	Australia	FI	Finland	ML	Mali
BB	Barbados	FR	France	MN	Mongolia
BE	Belgium	GA	Gabon	MR	Mauritania
BF	Burkina Faso	GB	United Kingdom	MW	Malawi
BG	Bulgaria	GN	Guinea	NL	Netherlands
BJ	Benin	GR	Greece	NO	Norway
BR	Brazil	HU	Hungary	PL	Poland
CA	Canada	IT	Italy	RO	Romania
CF	Central African Republic	JP	Japan	SD	Sudan
CG	Congo	KP	Democratic People's Republic of Korea	SE	Sweden
CH	Switzerland			SN	Senegal
CI	Côte d'Ivoire	KR	Republic of Korea	SU+	Soviet Union
CM	Cameroon	LI	Liechtenstein	TD	Chad
CS	Czechoslovakia	LK	Sri Lanka	TG	Togo
DE*	Germany	LU	Luxembourg	US	United States of America
DK	Denmark	MC	Monaco		

\+ Any designation of "SU" has effect in the Russian Federation. It is not yet known whether any such designation has effect in other States of the former Soviet Union.

PARSING PROGRAM DATA STREAMS

RELATED APPLICATIONS

This application is a continuation-in-part of commonly assigned U.S. Patent Application, Serial No. 07/537,466, filed June 11, 1990, for INTEGRATED SOFTWARE ARCHITECTURE FOR A HIGHLY PARALLEL MULTIPROCESSOR SYSTEM by George A. Spix et al.

FIELD OF THE INVENTION

The present invention relates to parsing a data stream having a plurality of different computer language syntax.

BACKGROUND OF THE INVENTION

Each programming language uses its own syntax and semantics; syntax used in the Fortran language is different from the C language syntax, etc. Programs written in any programming language have to be compiled, and during that process their syntax and semantics are verified. Syntax is the structure and specification of each language according to rules established for each language, i.e. grammar. Semantics of each language is the meaning conveyed by and associated with the syntax of such language. In compiling computer programs, parsing is an analysis of a stream of program expressions (sentences) for determining whether or not the program expressions are syntactically correct. Once it is determined that a stream of program expressions is syntactically correct, that steam of program expressions can be compiled into executable modules. Parsing is automatically performed in a computer using a computer program.

In parsing a computer program input stream, written in Fortran, for example, a scanner using a set of rules groups predetermined characters in the input stream into tokens. Scanners are programs constructed to recognize different types of tokens, such as identifiers, decimal constants, floating point constants, and the like. In recognizing or identifying a token, a scanner may look ahead in the input stream for additional characters.

The parser imposes a structure on the sequence of tokens using a set of rules appropriate for the language. Such rules are referred to as a context-free grammar; such rules are often specified in the so-called and well known Backus Naur form. Such a grammar specification for a programming language expression consisting of decimal digits and the operations + and "production" may be represented as follows:

```
E  :  E + T
E  :  T
T  :  T * F
T  :  F
F  :  decimal_digits
```

Each of the five grammar rules above, one on each line, is referred to a a "production". In the above program specification the tokens detected by the scanner are +,* and decimal-digits. Such tokens are passed to the parser program. Each string in the input stream that is parsed as having correct syntax is said to be "accepted". For example, the string 2+3*5 is "accepted" while the string 2++5 will be rejected as syntactically incorrect.

A left-to-right, right-most derivation(LR) parser accepts a subset of a context-free grammar. Each LR parser has an input, an output, a push-down stack, a driver program and a

parsing table. The parsing table is created from the grammar of the language to be parsed and is unique to such language and its grammar. The driver program serially reads tokens one at a time from the input stream. The input stream is typically stored in a computer storage and is scanned by the driver program scanning the stored input stream to fetch the tokens. Based upon the information in the parsing table that corresponds to the token being analyzed (input token) and the current program state, the driver program may shift the input token into the stack, reduce it by one of the productions, accept a string of such tokens, or reject the string of such tokens as being syntactically wrong. Reduction means that the right-hand side of a production is replaced by the left-hand side. An LR parser may also fetch a next token from the input stream for determining whether or not to shift or to reduce the token. Such a token is termed a "lookahead" token and is referred to herein as a look ahead portion of the input stream. The lookahead portion may include more than one token. When an LR parser performs reduction, additional semantic checks (also termed semantic actions) are performed.

A debugger, which may employ the above described parsing, is a software or program tool that assists a programmer to guide the execution of a program. It enables the programmer to monitor and modify the status of an executing program. A debugger usually can display values of variables, evaluate expressions, and display a procedure call chain at predetermined points in the executing program. A source level debugger allows a programmer to refer to information in a program in terms of the programming language in which the program was written An interactive debugger enables interactive operations between the debugger and the programmer. The programmer has to know the syntax and semantics of the debuggers command language. A debugger command might be PRINT expr where

expr is a programming language expression being evaluated with respect to the program being debugged. Known debuggers often define a syntax for expr that is independent of any programming language or is unique to one programming language. Since a programmer is familiar with the syntax and semantics of the programming language used to write the program being debugged, it makes the programmer's task easier if the same syntax were used to specify the expression the in PRINT command. Present day compiler technology permits the development of programs in which each module is written in a different programming language. Debugging such multi-language programs would be easier if the programmer could switch between different syntaxes during a debugging session. Then, to debug a multi-language program, switching parsing operations between the unique expression syntax may be required.

The command language and syntax of known debuggers have been designed in a variety of ways. Such designs have included language restrictions, unfamiliar debugger command languages and the lack of extensibility, i.e. adding another language to the debugger may be difficult or impossible. some debugger command languages are independent of any programming language. Such a selection may simplify the debugger but does not require each programmer to learn a new syntax to use the debugger. Another approach is to design a separate debugger for each programming language used and employ a syntax in the debugger command language that is similar to the respective syntax of the languages. While the programmer is not required to learn a new syntax, the debugger is necessarily limited to debugging programs written in one language. One could also design a debugger syntax that encompasses the syntax of all programming languages with which the debugger is expected to be used. Such a design is not language extensible. In these parsings systems, each token has but one meaning and interpretation irrespective of being used

for a debugger command language or for expression syntax of the programming language.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide a parsing method which is extensible to include additional programming languages in which the syntax of several languages is intermixed with a parsing command language and wherein like tokens in the input stream are capable of multiple meanings and interpretations.

Each of the plural programming languages and the debugger command language is parsed by a separate parser, each parser having its own parsing table and scanner. The command language parser initiates and controls the parsing including assigning parsing of programming language expressions to its respective parser. In one embodiment of the invention, the command language parser initiates parsing of each command and determines when the expression is to be parsed by which parser. The language parsers are called a a part of semantic actions performed by the command parser. All of the scanners operate with the same input stream. Each individual parser requests tokens only from its respective scanner. This selection of tokens results in diverse interpretations and meanings for tokens having identical character strings. Lookahead operations are handled, and include returning back to the input stream those lookahead portions of the input stream which will not be parsed by the parser that looked ahead. Selection of the language parsers can be explicit separately from the input stream or implicitly from the input stream. Parsing of syntax of the diverse programming languages and debugger commands is isolated by a plurality of separate parsers. The use of small parsing tables enhances the speed of parsing.

The above and other objects of the present invention will become apparent with reference to the appended drawings, the detailed description, and appended claims.

DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram of a prior art parsing system used in a program debugger.

Figure 2 is a simplified diagram of a parsing system using the present invention.

Figure 3 is a schematic showing the operations of the Figure 2 illustrated system.

DETAILED DESCRIPTION

Figure 1 illustrates a typical parsing system for a debugger. The input stream 10 contains debugger commands and programming-language specific expressions. In a parsing operation, the input stream is stored in a working data storage (not shown) of a data processing system (not shown). The scanner 11, a program, serially fetches the characters of the input stream 10 and groups these characters to identify tokens. The tokens are passed to debugger command language parser 12. Each identified token and other groups of characters in the input stream have but one meaning and interpretation for the command language parser 12.

In the present method, the three parsers (Figure 2), C language expression parser 25, Fortran language expression parser 23 and command language parser 16 are created using a well-known parser generator called YACC (Yet Another Compiler Compiler). The YACC driver program (skeleton of

the program) is modified to avoid global name conflicts. YACC enables semantic actions to be specified within grammar rules.

In Figure 2, input stream 10 is first accessed by scanner 15 for parsing. Scanner 15 is a debugger command language scanner for fetching and grouping characters from the input stream 10. Scanner 15 detected tokens are passed to command parser 1. Command parser 16 only parses the syntax for the command language of the debugger. The decision to use a particular language expression parser is based upon a debugger command termed the language command (see Table I below). This debugger command designates which programming language parser is to be used for parsing ensuing language-specific syntax in input stream 10. The programming language parser chosen by the language command remains in effect until a new language command is seen by the command parser 16. The language command is indicated by the LANGUAGE token while which one of the plurality of syntax is to be used is identified by the language command qualifier which. As shown in Table I, the qualifier which can indicate C, Fortran, or any future language. The extensibility also maintains consistency in language used by the programmer when debugging. This combination, plus the later-described token analysis for selecting a parser for a given programming language or command expression, provides enhanced capability and flexibility in parsers.

Also to be noted is the fact that the entire language command is parsed by the command parser 16. Each of the expressions written in diverse languages is parsed by a separate scanner-parser combination. This modular arrangement enables a relatively small parsing table to be used in each parser; small parsing tables increase parsing performance. Scanner 22 and Fortran parser 23 parse Fortran programming language expressions, scanner 24 and C parser 25 parse C language expressions while any additional

programming languages would be parsed by the combination of scanner 26 and future language parser 27. Additional programming language parsers can be added in a modular form as indicated by ellipsis 29. All of the scanner-parser combinations have a common access to input stream 10 via programming calls to or activations of the respective parsers 23, 25, 27 all as represented in Figure 2. When any of the parsers 23, 25, 27 complete the parsing, control is returned to the command parser 16.

Table I below includes an illustrative subset of a command language grammar. Each expr is a place holder for the expression grammar to be parsed by one of the programming language parsers 23, 25 or 27. The character strings BREAK, IF, THEN, ENDBREAK, PRINT, HEX, DECIMAL, LANGUAGE, C, FORTRAN, and \n are keywords or tokens. Source_line is a token that represents a source line at which to stop. The command language scanner 15 recognizes the tokens BREAK, IF, THEN, ENDBREAK, PRINT, HEX, DECIMAL, C, FORTRAN, LANGUAGE and source_line. When desired, the command language grammar can be modified to enable parsing of additional programming languages. Such language additions occur in the "which :" portion of Table I. For each new programming language added, a separate parser is designed for the language.

TABLE I

```
command_list    :    command_list command
                     command
                ;
command         :    BREAK source_line IF expr THEN
                          command ENDBREAK \ n
                     PRINT expr \n
                     LANGUAGE which \n
                ;
format          :    HEX
                     DECIMAL
                     /* empty */
                ;
which           :    C
                     FORTRAN
                ;
```

The command parser 16 acts differently in the case of a command token such as the IF token, which stands alone, and a command token like the PRINT token, which may be followed by another qualifying command token (HEX or DECIMAL). When the command language parser 16 recognizes that the tokens following each IF token are a part of the expression, it calls the currently active language expression parser. Since an expression token rather than a command token will follow the IF token, command parser 16 does not look ahead and consequently does not return the lookahead token to input stream 10. In the case of the PRINT token, command parser 16 looks ahead to determine whether or not the next following token is a HEX or a DECIMAL token. If the following token is not HEX nor DECIMAL, then the token following PRINT token is a first token of a language expression. In this instance, command parser 16 writes the lookahead token back into input stream 10 and calls the currently active language parser 23, 25 or 27 to parse the expression beginning with the token following the PRINT

token. When the lookahead token is either a HEX or DECIMAL token, the command parser 16 writes back the lookahead token (HEX OR DECIMAL) token, the command parser 16 writes back the lookahead token (HEX OR DECIMAL) and then calls the currently active language parser 23, 25 or 27 to parse subsequent expressions as a part of its semantic actions.

Table II below lists an illustrative subset for C language expression grammar. This subset grammar accepts expressions consisting of +, -, *, / and identifiers. Scanner 24 recognizes the +, -, /. * and c_identifier tokens.

TABLE II

```
expr                  :   additive_expr
                          error
                      ;
additive_expr         :   multiplicative_expr
                          additive_expr +
                               multiplicative_expr
                          additive_expr-_
                               multiplicative_expr
multiplicative_expr   :   unary_expr
                          multiplicative_expr * unary_expr
                          multiplicative_expr / unary_expr
unary_expr            :   * c_identifier
                      ;   c_identifier
```

An illustrative subset of grammar expressions for the Fortran parser 23 is shown below in Table III. This subset grammar accepts expressions consisting of +, -, *, /, and identifiers. Scanner 24 recognizes the tokens +, -, *, / and f_identifier.

TABLE III

```
expr                 :    additive_expr
                          error
                     ;
additive_expr        :    additive_expr +
                               multiplicative_expr
                          additive_expr - multiplicative_
                               expr
                               multiplicative-expr
multiplicative_expr  :    multiplicative_expr *
                               exponent_expr
                          multiplicative_expr
                               exponent_expr
                          exponent_expr
                     ;
exponent-expr        :    f_identifier ** exponent_expr
                          f_identifier
```

The scanners 22 and 24 use different rules for identifying

language parser then reduces the expression such as by the rule expr : additive_expr.

When any of the programming language parsers detect an error, the special error rule expr:error is used. This rule requires the parser 23, 25 or 27 to clear its stack (not shown) of tokens that constitute an expression. Execution of the error rule in an expression reduction is reported to command parser 16. In a constructed embodiment of the invention, after reporting the detected syntax error, command parser 16 finds the end-of-line marker \n and then resumes parsing at the first token beyond marker \n in data stream 10.

Figure 3 shows operation of the invention in a two-level parsing system having command parser 16 at a first or initial level and programming language parsers 23 and 25 in a second level. At the present level of description, parsers 23 and 25 appear to be identical, it being understood that the rules and token operations in the two parsers are different, as discussed above. In describing Figure 3, only parser 23 is described in detail, that description applies equally to parser 25.

The ensuing description assumes that the parsers 16 and 23 have been loaded into a computer, i.e. established for operation, the input stream 10 is in data storage mans, and the program is ready to execute. Scanner 15 obtains a first token from input stream 10. The first sequence of tokens is taken by command parser 16 for shifting and reduction at step 40. Any syntactical error is detected at branch step 41. Upon detecting an error, the error rule is invoked with the error reported via step 42 to the user (programmer) interface 43. The end of the current line is then located such that parsing beginning with the next line in input stream 10 can ensue. With no error being detected at step 41, command parser 16 determines at step 45 whether

or not the token following the examined token represents a programming language or is part of the command language. Such determination may require a lookahead in input stream 10 by parser 16. Upon successful reduction, the parsing of a command is complete. The cycle is started again for the next command in input stream 10. This processing continues until no more characters remain in input stream 10.

Command parser 16 calls one of the programming language parsers 23, 25 or 27, such as Fortran parser 23 as a part of its semantic actions. Fortran parser 23 invokes scanner 22. Scanner 22 serially fetches tokens from input stream 10. Fortran parser 23 then operates on a sequence of tokens until it recognizes a complete expression or encounters a syntax error. Such errors are reported to user interface 43. If there are no errors and Fortran parser 23 recognized a complete expression, then Fortran parser 23 writes back any lookahead token to input stream 10 and returns control to command parser 16. Upon retrieving control from Fortran parser 23, command parser 16 continues performing the shift/reduce operations until it successfully parses a command or encounters an error. This action completes parsing one command. If there are more characters in input stream 10, the entire described cycle is repeated. A token that is not parsed is written back to input stream 10 such that a next parser can read the token from the input stream 10 such that a next parser can read the token from the input stream via its scanner.

While the invention has been particularly shown and described with reference to preferred embodiments thereof, those skilled in the art will understand that various changes in form and details may be made therein without departing from the spirit and scope of the invention.

We claim:

1. In a machine-effected method of parsing a program input stream having a plurality of a machine-recognizable programming language indicators, each of the programming language indicators for indicating any one of a plurality of different expression syntax, including the machine-executed steps of:

establishing a plurality of sets of parsing syntax, one set for each of said different syntax, respectively;

detecting a one of the language indicators in the input stream and indicating which of the respective different syntax is to be used in parsing; and

selecting a parsing syntax indicated by the respective detected language indicator and then parsing its included programming language expressions using the selected parsing syntax.

2. In the machine-effected method set forth in claim 1 further including the machine-executed steps of:

in said indicating step, detecting in said input stream a language command having a one of said programming language indicators as a command qualifier, analyzing the detected language indicator, if the analyzed indicator indicates a first programming language, then performing said selecting step for selecting the first programming language syntax as a current syntax to use for ensuing programming language expressions; and

detecting in said input stream a second language command having a second programming language indicator as a command qualifier for indicating a second programming language syntax to use as a current syntax for ensuing programming language expressions.

3. In the machine-effected method set forth in claim 1 further including the machine-executed steps of:

while using said selected syntax for parsing, detecting a second programming language indicator which indicates a second syntax different from the selected syntax; and

parsing the ensuing program language expressions in said using said second syntax.

4. In the machine-effected method set forth in claim 2 further including machine-executed step of:

detecting a predetermined token in the input stream, looking ahead in the input stream for a next token which is a lookahead token, examining the next token;

when said examined token has a first indication, continue parsing with the then current syntax, otherwise selecting another syntax for parsing a predetermined sequence of tokens beginning with said predetermined token; and

before selecting the another syntax, returning the lookahead token to the input stream.

5. In the machine-effected method set forth i claim 2 further including the machine-executed steps of:

in said establishing step, embodying each of said sets of parsing syntax in a separate parser including establishing a separate scanner for each of the separate parsers, respectively;

establishing each of the separate scanners to recognize tokens for its respective parser and that each parser obtains tokens only from its own scanner;

establishing a one of the parsers to be an initial parser for performing said indicating and selecting steps for each of the programming language expressions; and

all of the established parsers excepting the initial parser upon completion of parsing a one of the programming language expressions activating the initial parser to perform said analyzing and selecting steps on a next programming language expression to be parsed.

6. In the machine-effected method set forth in claim 5 further including the machine-executed steps of:

when establishing said parsers, establishing in each of said individual parsers a set of parsing rules only for one command or programming language whereby parsing time is reduced compared to parsers including both command and programming languages in parsing tables.

7. In the machine-effected method set forth in claim 5 further including the machine-executed steps of:

while parsing any of the programming language expressions in any of the established parsers, detecting a syntax error and sending the detected syntax error from any of the parsers to one error logger.

8. In the machine-effected method set forth in claim 7 further including the machine-executed steps of:

creating a plurality of said parsers and storing same in a data storage means; and

initiating a parsing operation, when initiating the parsing operation performing all of said establishing steps first by selecting the parsers from the plurality of created parsers as the established parsers and then performing said analyzing and selecting steps on each programming language expression in the input stream.

9. In the machine-effected method set forth in claim 8 further including the machine-executed steps of:

selecting said initial parser to be a command language parser; and

selecting one or more of the created parsers other than said initial parser to include, respectively, parsing syntax for one or more programming languages.

10. In the machine-effected method set forth in claim 9 further including the machine-executed step of:

selecting one of the established parsers having a parsing syntax for FORTRAN or the C language.

11. In a machine-effected method of parsing an input stream of programming language expressions for detecting syntax errors in the expressions for enabling compiling the expressions into object code, including the machine-executed steps of:

receiving an input stream of programming language expressions having any one of a plurality of syntax;

establishing a plurality of parsing syntax; and

for each programming language expression, selecting a one of the parsing syntaxes for parsing each programming language expression which is the syntax of the respective programming language expressions.

12. In the machine-effected method set forth in claim 11 further including the machine-executed step of:

establishing a one of the parsing syntax to be a language syntax and another one of the parsing syntax to be a command syntax.

13. In the machine-effected method set forth in claim 12 further including the machine-executed steps of:

during said parsing while using a one of the parsing syntax as a current syntax, detecting a predetermined token that can be in any one of a plurality of syntax, then scanning ahead in the input stream to identify a next token; and

ascertaining the syntax using the next and predetermined tokens, if the ascertained syntax is then current parsing syntax then parsing the expression, otherwise returning the scanned ahead next token to the input stream and selecting a parsing syntax for the ascertained syntax.

14. In a machine-effected method of establishing an input stream for parsing, including the machine-executed steps of:

establishing a plurality of sets of syntax, each set being for a predetermined set of rules different from rules in the other sets, establishing character strings using said different rules such that predetermined identical character strings represent different meanings in the respective sets of syntax; and

creating an input stream including programming language expressions written in any one of a plurality of programming languages and inserting before a sequence of programming language expressions written in a programming language different from a programming language used to write preceding programming language expressions a language command indicating a new programming language and its syntax are to be used in parsing the sequence of programming language expressions such that the input stream explicitly indicates the respective syntax to be used in parsing ensuing programming language expressions and interleaving the programming language expressions having the different syntax in the input stream.

15. In apparatus for parsing an input stream of programming language expressions having programming language expressions using diverse syntax, including, in combination:

a plurality of syntax means each adapted to receive an expression from the input stream for parsing, one syntax means for each of the diverse syntax and having means for parsing the respective diverse syntax, respectively; and

analyzing means for receiving the input stream for detecting in the input stream which of the respective diverse syntax is used in ensuing programming language expressions and being connected to the plurality of syntax means for actuating a respective syntax means to parse the programming language expression having the respective syntax.

16. In the apparatus set forth in claim 15, further including, in combination:

token means in the analyzing means for detecting tokens in said programming language expressions and for analyzing the detected tokens for indicating a first and second type of detected tokens;

selecting means in the analyzing means for responding to the token means indicating a first type of detected token to select a first one of the syntax means for parsing the expression in which the detected token resides and responding to the token means indicating a second type of detected token to select a second one of the syntax means for parsing the expression in which the detected token reside.

17. In the apparatus set forth in claim 16 further including, in combination:

said token means including lookahead means for fetching and analyzing additional tokens in the input stream in addition to a one token being analyzed and including means for returning the additional tokens to the input stream before selecting said second one syntax means and not returning the additional tokens when selecting said first one syntax means.

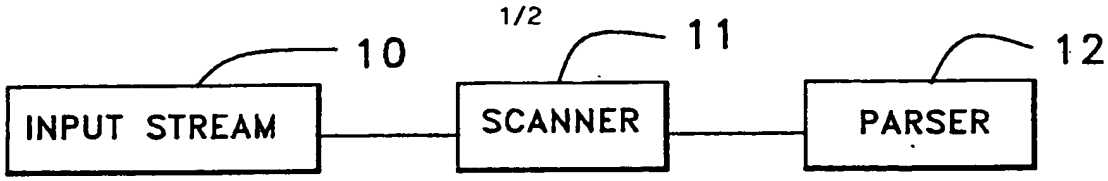


FIG.1 PRIOR ART

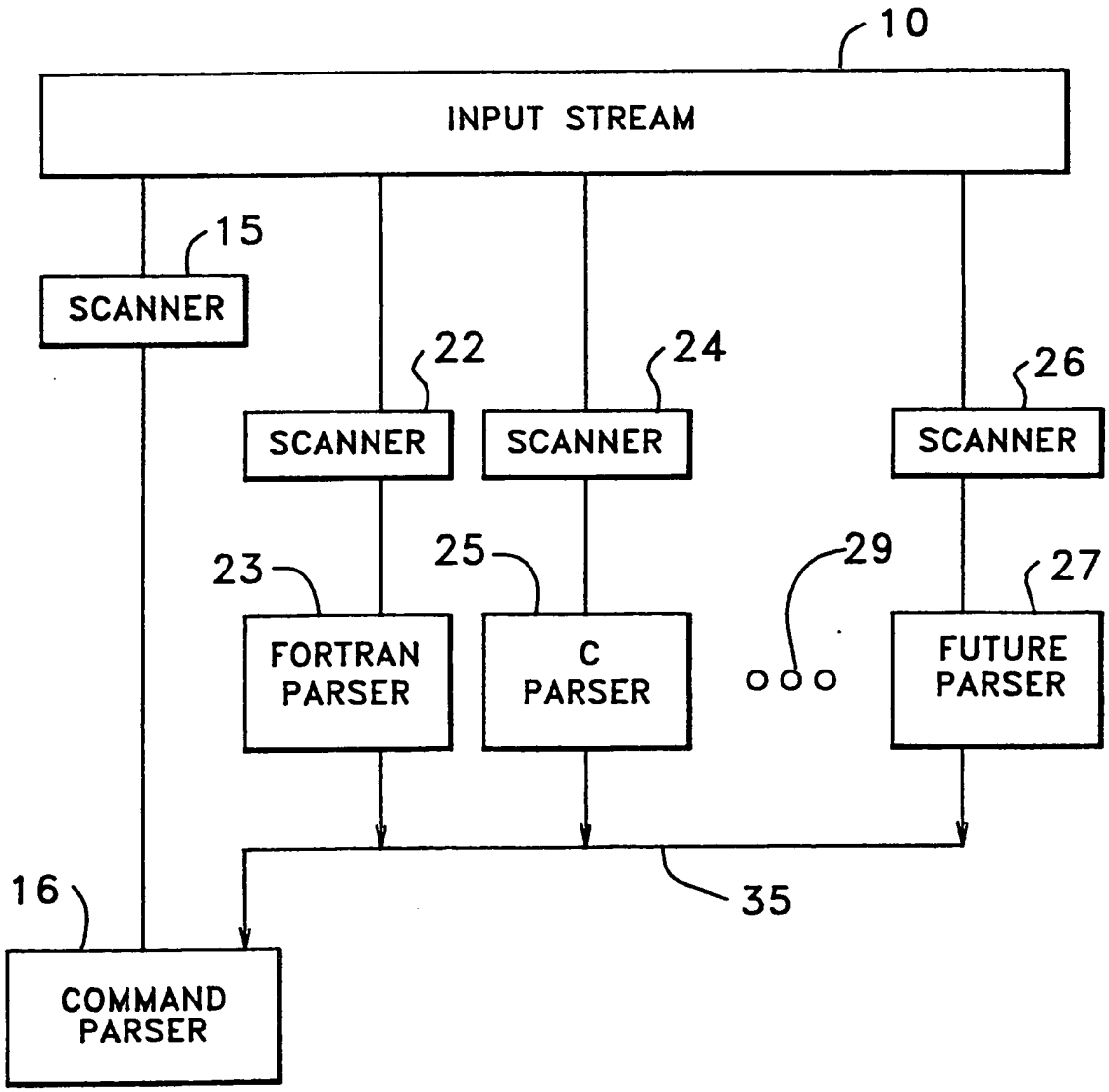


FIG.2.

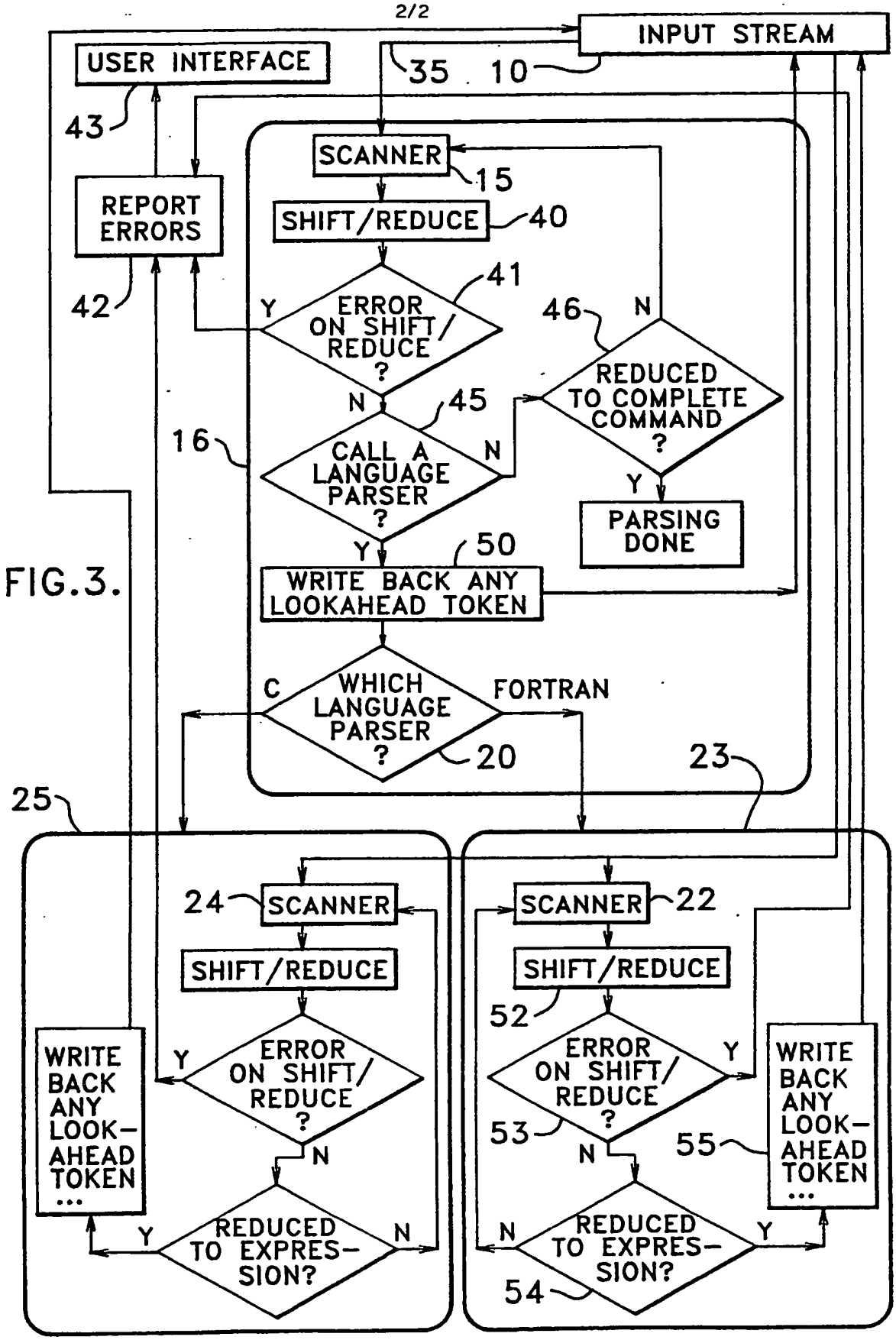

INPUT STREAM
USER INTERFACE
35
10
43
SCANNER
15
REPORT ERRORS
SHIFT/REDUCE
40
41
42
Y
ERROR ON SHIFT/ REDUCE ?
46
N
REDUCED TO COMPLETE COMMAND ?
N
45
16
CALL A LANGUAGE PARSER ?
N
Y
PARSING DONE
Y
50
WRITE BACK ANY LOOKAHEAD TOKEN
C
WHICH LANGUAGE PARSER ?
FORTRAN
20
23
25
24
SCANNER
SCANNER
22
SHIFT/REDUCE
SHIFT/REDUCE
52
WRITE BACK ANY LOOK-AHEAD TOKEN ...
Y
ERROR ON SHIFT/ REDUCE ?
ERROR ON SHIFT/ REDUCE ?
Y
53
55
WRITE BACK ANY LOOK-AHEAD TOKEN ...
N
N
Y
REDUCED TO EXPRES-SION?
N
REDUCED TO EXPRES-SION?
Y
54

FIG.3.

INTERNATIONAL SEARCH REPORT

International Application No. PCT/US91/04071

I. CLASSIFICATION OF SUBJECT MATTER (if several classification symbols apply, indicate all) [6]

According to International Patent Classification (IPC) or to both National Classification and IPC

IPC (5): G06F 9/45
U.S.CL.: 395/70

II. FIELDS SEARCHED

Minimum Documentation Searched [7]

Classification System	Classification Symbols
U.S.CL.	364/200,900

Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched [8]

III. DOCUMENTS CONSIDERED TO BE RELEVANT [9]

Category *	Citation of Document, [11] with indication, where appropriate, of the relevant passages [12]	Relevant to Claim No. [13]
A	US, A, 4,686,623 (WALLACE) 11 AUGUST 1987 (SEE THE ABSTRACT)	1-17
A	US, A, 4,833,606 (IWASAWA) 23 MAY 1989 (SEE THE ABSTRACT)	1017
A	US, A, 4,885,684 (AUSTIN) 05 DECEMBER 1989 (SEE THE ABSTRACT)	1-17

* Special categories of cited documents: [10]

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

IV. CERTIFICATION

Date of the Actual Completion of the International Search

05 DECEMBER 1991

Date of Mailing of this International Search Report

08 JAN 1992

International Searching Authority

ISA/US

Signature of Authorized Officer

For DAVID Y. ENG NGUYEN NGOC-HO
INTERNATIONAL DIVISION

Form PCT/ISA/210 (second sheet) (Rev.11-87)